shell 脚本读取命令行的参数

前提

在编写 shell 程序时经常需要处理命令行参数

选项与参数：

如下命令行：

```
./test.sh -f config.conf -v --prefix=/home
```

```
-f为选项，它需要一个参数，即config.conf,
-v 也是一个选项，但它不需要参数。
--prefix我们称之为一个长选项，即选项本身多于一个字符，它也需要一个参数，用等号连接，当然等号不是必
/home可以直接写在--prefix后面，即--prefix/home,
更多的限制后面具体会讲到。
```

一. 手工处理方式 (已验证)

在手工处理方式中，首先要知道几个变量，还是以上面的命令行为例：

```
$0:  ./test.sh,即命令本身，相当于c/c++中的argv[0]
$1:  -f,第一个参数.
$2:  config.conf
$3, $4 ... : 类推。
$#: 参数的个数，不包括命令本身，上例中$#为4.
$@: 参数本身的列表，也不包括命令本身，如上例为 -f config.conf -v --prefix=/home
$*: 和$@相同，但"$*" 和 "$@"(加引号)并不同，"$*"将所有的参数解释成一个字符串，而"$@"是一个参数数组
```

例子

```
#!/bin/bash
for arg in "$*"
  do
    echo $arg
  done
for arg in "$@"
  do
    echo $arg
  done
```

执行./test.sh -f config.conf -n 10 会打印：

```
# 这是"$*"的输出
-f config.conf -n 10

#以下为$@的输出
-f
config.conf
-n
10
```

所以，手工处理的方式即对这些变量的处理。因为手工处理高度依赖于你在命令行上所传参数的位置，所以一般都只用来处理较简单的参数。

例如：

```
./test.sh 10
```

而很少使用./test -n 10 这种带选项的方式。

典型用法为：

```
#!/bin/bash
if [ x$1 != x ]
then
  #...有参数逻辑
else
then
  #...没有参数逻辑
fi
```

为什么要使用 x$1 != x 这种方式来比较呢？(x 就是任意的一个字符, 也可以是别的)

想像一下这种方式比较：

```
if [ -n $1 ]  #$1不为空
但如果用户不传参数的时候，$1为空，这时就会变成 [ -n ] ,所以需要加一个辅助字符串来进行比较。
```

手工处理方式能满足大多数的简单需求，配合 shift 使用也能构造出强大的功能，但在要处理复杂选项的时候建议用下面的两种方法。

二. getopts/getopt

处理命令行参数是一个相似而又复杂的事情，为此，c 提供了 getopt/getopt_long 等函数，

c++ 的 boost 提供了 options 库，在 shell 中，处理此事的是 getopts 和 getopt.

getopts 和 getopt 功能相似但又不完全相同，其中 getopt 是独立的可执行文件，而 getopts 是由 bash 内置的。

```
./test.sh -a -b -c  ：  短选项，各选项不需参数
./test.sh -abc   ：  短选项，和上一种方法的效果一样，只是将所有的选项写在一起。
./test.sh -a args -b -c ：短选项，其中-a需要参数，而-b -c不需参数。
./test.sh --a-long=args --b-long ：长选项
```

先来看 getopts, 它不支持长选项。

使用 getopts 非常简单：

```
#test.sh
#!/bin/bash
while getopts "a:bc" arg #选项后面的冒号表示该选项需要参数
do
        case $arg in
             a)
                echo "a's arg:$optarg" #参数存在$optarg中

             b)
                echo "b"

             c)
                echo "c"
```

```
            ?)  #当有不认识的选项的时候arg为?
                echo "unkonw argument"
        exit 1

        esac
done
```

现在就可以使用:

```
./test.sh -a arg -b -c
```

或

```
./test.sh -a arg -bc
```

来加载了。

应该说绝大多数脚本使用该函数就可以了，如果需要支持长选项以及可选参数，那么就需要使用 getopt.

getopt 自带的一个例子:

```
#!/bin/bash
# a small example program for using the new getopt(1) program.
# this program will only work with bash(1)
# an similar program using the tcsh(1) script language can be found
# as parse.tcsh
# example input and output (from the bash prompt):
# ./parse.bash -a par1 'another arg' --c-long 'wow!*\?' -cmore -b " very long "
# option a
# option c, no argument
# option c, argument `more'
# option b, argument ` very long '
# remaining arguments:
# --> `par1'
# --> `another arg'
# --> `wow!*\?'
# note that we use `"$@"' to let each command-line parameter expand to a
# separate word. the quotes around `$@' are essential!
# we need temp as the `eval set --' would nuke the return value of getopt.
#-o表示短选项，两个冒号表示该选项有一个可选参数，可选参数必须紧贴选项
#如-carg 而不能是-c arg
```

```
#--long表示长选项
#"$@"在上面解释过
# -n:出错时的信息
# -- ：举一个例子比较好理解：
#我们要创建一个名字为 "-f"的目录你会怎么办？
# mkdir -f #不成功，因为-f会被mkdir当作选项来解析，这时就可以使用
# mkdir -- -f 这样-f就不会被作为选项。
temp=`getopt -o ab:c:: --long a-long,b-long:,c-long:: \
     -n 'example.bash' -- "$@"`
if [ $? != 0 ] ; then echo "terminating..." >&2 ; exit 1 ; fi
# note the quotes around `$temp': they are essential!
#set 会重新排列参数的顺序，也就是改变$1,$2...$n的值，这些值在getopt中重新排列过了
eval set -- "$temp"
#经过getopt的处理，下面处理具体选项。
while true ; do
        case "$1" in
                -a|--a-long) echo "option a" ; shift ;;
                -b|--b-long) echo "option b, argument \`$2'" ; shift 2 ;;
                -c|--c-long)
                        # c has an optional argument. as we are in quoted mode,
                        # an empty parameter will be generated if its optional
                        # argument is not found.
                        case "$2" in
                                "") echo "option c, no argument"; shift 2 ;;
                                *)  echo "option c, argument \`$2'" ; shift 2 ;;
                        esac ;;
                --) shift ; break ;;
                *) echo "internal error!" ; exit 1 ;;
        esac
done
echo "remaining arguments:"
for arg do
   echo '--> '"\`$arg'" ;
done
```

比如使用

```
./test -a -b arg arg1 -c
```

你可以看到, 命令行中多了个 arg1 参数，在经过 getopt 和 set 之后，命令行会变为:

```
-a -b arg -c -- arg1
```

1 指向 – a, 2 指向 - b,3 指向 arg,4 指向 - c,$5 指向 --, 而多出的 arg1 则被放到了最后。

三. 总结

一般小脚本手工处理也就够了，getopts 能处理绝大多数的情况，getopt 较复杂，功能也更强大。

站在巨人肩膀上摘苹果

https://www.jb51.net/article/48686.htm

全文完

本文由 简悦 SimpRead 转码，用以提升阅读体验，原文地址